CS 211	Computers and Programming Spring 2002	
	Prelim II Solutions	**April 16th, 2002**

NAME:____

CU ID:________________________________

Recitation instructor/time________________________________

You have one and a half hours to do this exam.

All programs in this exam must be written in Java. Excessively convoluted code will not be graded.

****

Problem	Score
1	
2	
3	
4	
5	
6	
Total	

1. (15 points)

   (a) (10 points) Write a Java class method named *append* that takes two *non-empty* linked lists `L1` and `L2` as parameters, and updates the last cell of `L1` so that it points to the first cell of `L2`, as shown in Figure 1 below. The method does not return anything. The ListCell class from lecture is reproduced at the end of the exam. You may NOT use the Java LinkedList class.

   (b) (5 points) What is the asymptotic complexity of your algorithm, expressed as a function of n_1 and n_2 where n_1 is the number of elements in `L1`, and n_2 is the number of elements of `L2` respectively? Justify your answer briefly.

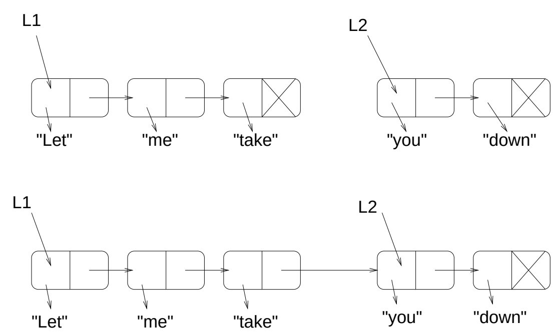


Figure 1: Appending two lists

(a)

```
public static void append(ListCell L1, ListCell L2) {

  ListCell finger = L1;

  while (finger.getNext() != null)
        finger = finger.getNext();

  finger.setNext(L2);

}
```

(b) This is an $O(n_1)$ algorithm because the number of iterations of the while-loop is equal to n_1, and each iteration performs a constant amount of work.

2. (15 points)

   Suppose $f_1(n) = O(g(n))$ and $f_2(n) = O(g(n))$. Answer the following questions.

   (a) Consider the function $h(n) = f_1(n) + f_2(n)$. Is $h(n) = O(g(n))$? Justify your answer formally using witness pairs (k, N) as described in class.

   (b) Consider the function $k(n) = f_1(n) * f_2(n)$. Is $k(n) = O(g(n))$? Justify your answer formally.

   (a) Yes.

   We know that there exist c_1, n_1, c_2 and n_2 such that
   $f_1(n) \leq c_1 * g(n)$ for all $n \geq n_1$
   $f_2(n) \leq c_2 * g(n)$ for all $n \geq n_2$
   Let $c_s = c_1 + c_2$ and $n_s = max(n_1, n_2)$. It is easy to see that for all $n > n_s$, $f_1(n) + f_2(n) \leq c_s * g(n)$.

   (b) No. Let $f_1(n) = n$ and $f_2(n) = n$.

   So $f_1(n) = O(n)$ and $f_2(n) = O(n)$.
   We will show that $k(n) = n^2 \neq O(n)$. If not, we can find a c and n_0 such that for all $n > n_0$, $n^2 \leq c * n$. But this is impossible becaue $(n^2 - c * n) < 0$ implies that $(n - c) < 0$, so $n < c$ which contradicts the assumption that this holds for all $n > n_0$.

3. (30 points)

(a) (10 points) Write a recursive class method that computes the number of nodes in a binary tree. Assume that the root of the tree is a parameter to the method. The TreeCell class from lecture is reproduced at the end of the exam.

(b) (2 points) Does your method do an in-order, post-order, or pre-order walk of the tree?

(c) (3 points) What is the asymptotic complexity of your method? Explain your answer briefly.

(d) (10 points) Write a recursive class method to print the values stored in a binary search tree. You may assume that the root of the tree is a parameter to the method. The values must be printed in *descending* order — that is, the largest value must be printed first. You may assume that each data item has a *toString* method that returns a string representation of that item.

(e) (2 points) Does your method do an in-order, post-order, or pre-order walk of the tree?

(f) (3 points) What is the asymptotic complexity of your method? Explain your answer briefly.

(a)
```
public static int numNodes(TreeCell t) {
    if (t == null) return 0;
    else return 1 + numNodes(t.getLeft())
                  + numNodes(t.getRight());
}
```

(b) Post-order traversal

(c) Complexity $= O(n)$ where n is number of nodes in tree.

(d)
```
public static void printValues(TreeCell t) {
  if (t == null) return;
  else {
        printValues(t.getRight());
        System.out.print(" " + t.getDatum() + " ");
        printValues(t.getLeft());
  }
}
```

(e) In-order traversal

(f) $O(n)$ where n is the number of nodes in the tree.

4. (Short answers) (20 points)

Time in the following questions refers to *worst-case asymptotic complexity.*

(a) Linear search of an array requires that the array be sorted. True or false.

(b) Linear search in a sorted array of n elements takes time ________.

(c) Binary search in an array requires that the array be sorted. True or false.

(d) Binary search in a sorted array of n elements takes time ________.

(e) Quick-sort of an array of n elements takes time ________.

(f) Merge-sort of an array of n elements takes time ________.

(g) Insertion into a sorted list of n elements takes time ________.

(h) Deletion from a sorted list of n elements takes time ________.

(i) Search in a (not necessarily balanced) binary search tree of n elements takes time ________.

(j) Deletion in a (not necessarily balanced) binary search tree of n elements take time ________.

(a) False

(b) $O(n)$

(c) True

(d) $O(log(n))$

(e) $O(n^2)$

(f) $O(nlog(n))$

(g) $O(n)$

(h) $O(n)$

(i) $O(n)$

(j) $O(n)$

5. (12 points) A binary tree is known to have $2^n - 1$ nodes where n is some positive integer greater than 1, but nothing else is known about its structure.

   (a) What is the smallest number of leaf nodes it can have?

   (b) What is the largest number of leaf nodes it can have?

   (c) What is the largest number of edges that can be there in a simple path from the root of the tree to a leaf?

   (d) What is the smallest number of edges that can be there in a simple path from the root of the tree to a leaf?

   (a) Smallest number of leaf nodes = 1. Example: chain of nodes.

   (b) Largest number of leaf nodes = 2^{n-1}. Example: complete tree.

   (c) Largest number of edges = $2^n - 2$. Example: chain of nodes.

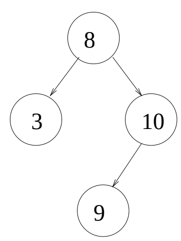
   (d) Smallest number of edges = 1. Example: single node hanging off root and other nodes in other sub-tree of root.

6. (8 points) We know that a sorted array can contain values in either ascending or descending order. As defined in class, binary search trees contain values in "ascending" order — that is, the left sub-tree of a node contains values less than the value at the node, while the right sub-tree contains values greater than the value at that node.

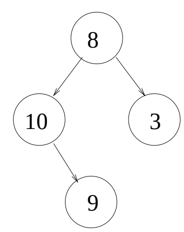
   Define a *reverse* binary search tree to be a binary search tree that contains values in "descending" order — that is, the left sub-tree of a node contains values greater than the value at that node, while the right sub-tree contains values less than the value at that node.

   (a) Write a recursive class method to modify a binary search tree into the corresponding reverse binary search tree. Assume that the root of the tree is passed as a parameter to the method, and that the tree is represented using the TreeCell class given at the end of this exam.

   (b) Does your method perform an in-order, post-order or pre-order walk of the tree?



Binary Search Tree



Reverse Binary Search Tree

(a)
```
public static void reverseBST(TreeCell t) {
   if (t == null) return;
   //swap left and right sub-trees
   TreeCell temp = t.getLeft();
   t.setLeft(t.getRight());
   t.setRight(temp);
   //recursively handle sub-trees
   reverseBST(t.getLeft());
   reverseBST(t.getRight());
}
```

```
//Another solution is to reverse sub-trees first, and then swap
//left and right sub-trees
```

(b) Pre-order (alternative solution is post-order)